proceedings of C# and .NET Technologies'2003 Int.Workshop, ISBN 80-90301-3-6, UNION Agency – Science Press, Plzen, Czech Republic, pp.87-92, 2003

The Implicit Function Modeling System - Comparison of C++ and C# Solutions

Uhlir Karel[1]
University of West Bohemia
Univerzitni 8
30614 Plzen, Czech Republic
kuhlir@kiv.zcu.cz

Skala Vaclav
University of West Bohemia
Univerzitni 8
30614 Plzen, Czech Republic
skala@kiv.zcu.cz

ABSTRACT

This paper compares possibilities of Visual .NET C# with other compiled programming environments (e.g. Microsoft Visual C++, Borland C++ Builder) and interpreted languages like for example HyperFun. The study focuses on the aspects of developing a system for direct interpretation of objects described by implicit functions. The starting point of the comparison is based on a HyperFun system. The results show the speed-up reached by compiling implicitly defined objects and illustrate possibilities of tested languages to create similar structures.

Keywords

Implicit function, C#, C++, modeling system, objects.

1. Introduction

This paper presents a system developed for direct compiling of implicitly defined objects. There exist many systems for modeling with implicitly defined objects. One of them is the HyperFun (HF) system. This system uses HyperFun language for a model description. HF performs interpretation of a model to its internal structures. The HyperFun language is very similar to C programming language. The new Compiled HyperFun (CHF) system is introduced in this article. The CHF system performs direct compilation of the model. There was a tendency to have the same models for the CHF and the HF. The CHF was designed to process identical programming and data structures Therefore, overloaded operators were used to accomplish it. The direct compilation of the object offers significant speed-up. The speed-up tests were created with programming languages mentioned above. We have selected C++ and C# languages for our experiments. The comparison of speed-up for C++ using Borland C++ Builder and Microsoft Visual C++, and C# programming language using Microsoft Visual C# (Microsoft Visual Studio .NET) made. The comparison of the approaches (HP and CHF) languages is presented using structures with an identical functionality. The CHF system is written in C# without using any special feature of C#.

1st Int.Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing

February 6-8, 2003, Plzen, Czech Republic.

ISBN 80-903100-3-6

2. Implicitly defined objects

The implicitly defined objects are objects described by an implicit function. Implicit functions define an implicit surface. An implicit function is a continuous scalar-valued function over the domain $\mathbf{R}^3$. The implicit surface of such a function is the locus of points at which the function takes the zero value. For example a unit sphere may be defined using the implicit function $f(x) = 1 - x^2 - y^2 - z^2$. Locations of points on the sphere are defined by the function $f(x) = 0$. Positive values are inside the object defined by the implicit function and outside of it there are negative values. This will be convention in this paper.

Basic operations used on implicitly defined objects are the Boolean operations. Using these operations can create a CSG tree. The CSG tree structure is one of basic methods used for description of structured models with primitives. In basic operations there behoove: union, intersection, subtraction and negation. The definition of these operators can be different [Pasko95]. It depends on geometric

[1] project supported by the Ministry of Education of The Czech Republic – Project MSM 235200005 and Microsoft Research Ltd.(U.K.)

continuity, which is to be achieved. The basic expression of these operations that are continuous is C^0 (Fig.1) is used. This operator's expression is very convenient for calculation. It is just the comparison of values.

Union	$F(x) = \max(f_1(x), f_2(x))$
Intersection	$F(x) = \min(f_1(x), f_2(x))$
Subtraction	$F(x) = \min(f_1(x), -f_2(x))$
Negation	$F(x) = -f(x)$

Figure 1. Operations definition.

Several methods for obtaining the surface from the function definition exist [Cerm02a, Cerm02b, Bloo94]. There is used a classic marching cubes method (Fig.2). The HF uses different method of iso-surface extraction but the evaluation part is identical. For both methods it is necessary to evaluate the function in each intersection of the regular grid. The density of the grid defines the quality of a final surface. Time required for the model evaluation in each intersection of the regular grid was the basic indicator of the method speed-up.

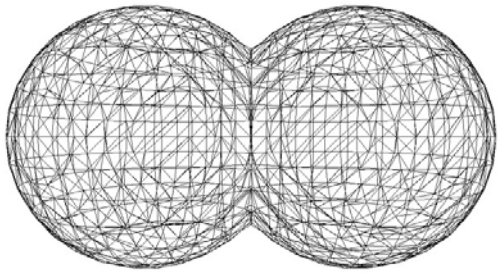

Figure 2. Union of two spheres.

3. HyperFun

The HyperFun is a simple geometric modeling language developed at the University of Hosei [Adz99]. It is intended for modeling of geometric objects described by an implicit function. This software tool uses a high-level programming language called HyperFun for description of models. The HyperFun has got a library of models. This library contains definitions of primitives, operations, and relations used in HyperFun models. A HyperFun model (Fig.2) is a text file, of a defined structure (Fig.3). The HyperFun is a system with Symbolic user interface (SUI) form modeling, parsing the model to internal data structures, evaluating the model and its visualization.

```
my_model(x[3], a[1])
{
 array center[3];
 center = [5,5,0];
 sp1=hfSphere(x,center,4.);

 center[2] = -5;
 sp2=hfSphere(x,center,4.);

 my_model = sp1 | sp2;
}
```

Figure 3. Model structure.

A HyperFun model can contain specifications of several geometric objects. Each object is defined by a function that is parametrized by input arrays of point coordinates and free numerical parameters. A model in HF can be more complex. There can be used auxiliary local variables and arrays, conditional sections (if-then-else), and iteratives ('while-loop'). Functional expressions are built using conventional arithmetic and relational operators. It is possible to use standard mathematical functions ('exp', 'log', 'sqrt', 'sin', etc.). Fundamental set-theoretic operations are supported by special built-in operators with reserved symbols ("|" - union, "&" - intersection, "\" - subtraction, "~" - negation, "@" - Cartesian product) [Pasko95, Rigau99].

The HF interpreter provides parsing with a syntax and semantic analysis. For each object described in the (HF) program, the function generates an internal representation. Accordingly, the internal representation can be considered as a tree structure ready to evaluate effectively all expressions defining the whole function. The internal representation of the object serves as an input parameter for the function that returns the value of the evaluated function at the given point. Visualization of a final scene is performed by the ray-marching algorithm [Pasko93].

4. Compiled HyperFun

The Evaluation is a computationally very expensive part of the HF, so the Compiled HyperFun (CHF) is used to speed-up this particular area. The idea of the CHF was to make identical system, which can take advantage of a classic compiler for a model evaluation.

As we intended to have the same language construction in both systems, the CHF had to be developed with this restriction. It was important to keep the same structure for the HF and the CHF, because it guarantees usability for the HF users.

```
double CP::fSphere (double x[],double
ct[],double r)
{
 double mx = x[0] - ct[0];
 double my = x[1] - ct[1];
 double mz = x[2] - ct[2];
 double x2 = mx*mx;
 double y2 = my*my;
 double z2 = mz*mz;
 double r2 = r*r;

 return (r2 - x2 - y2 - z2);
}
```

Figure 4. Sphere primitive implementation.

As it was already said, the basic functions are the same. Now it is time to introduce which way of the HF and the CHF are used for modeling. Where is the power of these systems? The both systems have a library of functions. This library can be divided into two parts. The first part contains basic primitives and the second one operations.

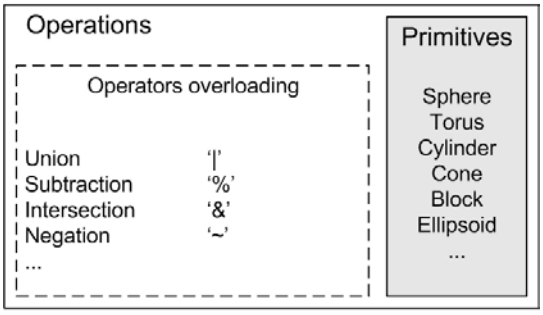

Figure 5. Inheritance.

The library of primitives includes objects e.g. sphere, torus, cylinder, ellipsoid, block, etc. All primitives have parameters which must be defined before the model evaluation is started. Each primitive returns the value for given parameters (e.g. spatial coordinates, diameter etc.) (Fig.4). As an example consider a sphere model. The sphere primitive is then calculated in the model evaluation. The primitive returns the value in spatial coordinates (x[]) and parameters are spatial coordinates of the center point (ct[]) and radius of the sphere (Fig.4). Each model contains input parameters. Model parameters are spatial coordinates and free parameters. The free parameters must not be defined (Fig.3). The model returns the value of the model in spatial coordinates.

The library of operations contains e.g. operators (union, intersection, subtraction, negation, etc.), blending operations [Pasko95], rotation etc. The operations are generally unary or binary. Their parameters are primitives.

```
FD& FD::operator |(FD &pA)
{
 if(this->m_dRes < pA.m_dRes)
  return(pA);
 else return(*this);
}
```

Figure 6. Union operator implementation in C++.

```
public static FD operator |(FD
first,FD second)
{
 if(first.m_dRes < second.m_dRes)
  return(second);
 else return(first);
}
```

Figure 7 . Union operator implementation in C#.

Object oriented programming languages C++ and C# were used for implementation of the library. There was implemented a class for primitives (CP on Fig.4) and a class for operations (FD on Fig.6). The class for operations was inherited from the class of primitives (Fig.5). This inheritance of classes was necessary for the construction of objects in C++ (Fig.6) and in C# (Fig.7), too. The implementation of library elements is almost identical in the both languages.

The main reason for the inheritance of the classes was the possibility of overloading operators. If we look at the model structure (Fig.3), it is clear, that characters are used for the basic HF operations and they are also operators in languages C++ or C#. Without the overloading operator it would not be possible to reach the same structure of model description like in the CHF. The problem was the usage of non-standard operators in the HF for some operations. These operators cannot be overloaded in C++ or C#. In particular, are speaking about the subtraction operator. In HF, the backslash operator ('\') is used for this operation. This operator cannot be overloaded, in the CHF the operator '%' is used for this operation. In C# language, there is also another problem. The equality operator (=) also cannot be overloaded. It slightly increased complexity of the model description.

```
FD FD::sphere_union(double x[])
{
 double center[]= {5.0,5.0,0.0};
 FmodelDouble  sp1,sp2;
 sp1 = fSphere(x,center,4.0);

 center[1]= -5.0;
 sp2 = fSphere(x,center,4.0);

 return(sp1 | sp2);
}
```

Figure 8. Model description in C++ language.

The paragraph above described main problems with the library of primitives and the operations. Other issues are clear from the model description in C++ (Fig.8) or C# (Fig.9) language. There are no significant differences of the model description in the C++ implementation and the model description in the HF.

For the reason of C++ or C# language conventions, there are some small differences in the model description, namely in the definition of variables. These differences cannot argue out potential users from using the CHF if they know the HF convention.

```
public FD sphere_union(double[] x)
{
 double[] center = {5.0,5.0,0.0};
 FD sp1 = new ClassOperator();
 FD sp2 = new ClassOperator();

 sp1.m_dRes = fSphere(x,center,4.0);

 center[1] = -5.0;
 sp2.m_dRes = fSphere(x,center,4.0);

 return(sp1 | sp2);
}
```

Figure 9. Model in C# language.

All the important differences were defined and now the evaluation of the model can start. Note, that the main interest of this work is the evaluation part only. Libraries and models together compose a project. The project must be compiled and the final result of the compilation is an executable file.

5. Results

The new method based on the direct compilation of models is working and new questions are arising now. Which compiler has got the best code optimization? Does the setting of a compiler affect the final evaluation time? These questions will be answered in next paragraphs.

There exist many different C++ language compilers and few C# compilers. As this work was oriented towards Microsoft Windows environment, basic testing was run with the most popular compilers for this operating system. Testing was performed in Microsoft Visual C++, Microsoft .NET C# and Borland C++ Builder compilers. C++ programming environments were selected for better idea about compiler optimizations and C# language for comparing with non-full object oriented language like C++. Finally the changing of the compiler setting was tested too but these changes did not have any important effect.

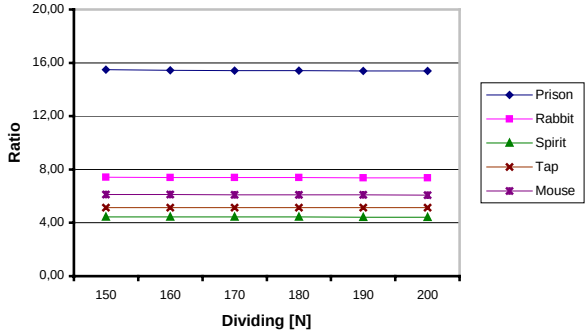

Figure 10. Speed-up ratio (Microsoft Visual C++).

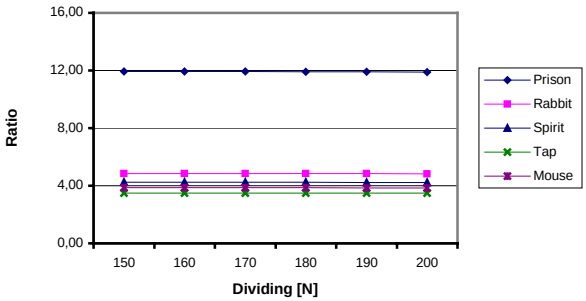

Figure 11. Speed-up ratio (Borland C++ Builder).

As it was already mentioned, the testing was done with the regular grid and the function was evaluated at each intersection of this grid. The space dividing was changed on the interval from 150 cells in one axis (N^3 cells) to 200 cells. All tests were run on the two-processor computer Pentium III, 450MHz, 1GB RAM, Windows XP.

All the graphs express the speed-up ratio of the tested model in the defined environment and language compared to the same model evaluated in the HF. The calculate time of the model in the HF divided by

the calculate time of the identical model in defined language and environment is the speed-up ratio. It can be seen in graphs that the highest speed-up was reached with Microsoft Visual C++ (Fig.10). The evaluation in this environment was much faster than in Borland C++ Builder each time (Fig.11). There were no differences between sources in Microsoft Visual C++ and Borland C++ Builder. The speed-up ratio of the model implemented in C# lies between both implementations in C++. The main reason is full object oriented approach and its cost in terms of the speed. Finally, it is clear that there is a considerable speed-up of compiled models (Fig.13) in comparison to the HF model.

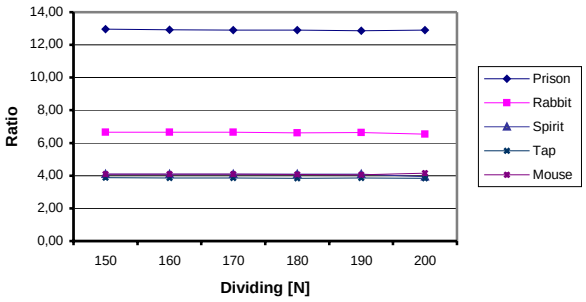

Figure 12. Speed-up ratio (Microsoft .NET C#).

Compiler	Min speed-up	Max speed-up	Average speed-up
B. C++ Builder	3,49 (Model 4)	11,92 (Model 1)	4,72
MS Visual C++	4,44 (Model 3)	15,43 (Model 1)	6,41
MS .NET C#	3,86 (Model 4)	12,91 (Model 1)	5,26

Table 1. Speed-up table.

6. Conclusion

This article showed a new method for the model evaluation that is based on a classic compiler. The method brought the speed-up for the model evaluation (Table 1). The comparison of classical compilers and the HF performed on the tested models showed differences in terms of the speed-up. Comparing C++ with C# shows what a user can expect from the programming language changes. The model rewritten in to the compiled programming language allowed all possible programming structures of the language like e.g. recursion. By using the overloading of operators, the identical description of the model as in the HF language was achieved.

7. References

[Adz99] Adzhiev, V., Cartwright, R., Fausett, E., Ossipov, A., Pasko, A., Savchenko, V.: HyperFun project: Language and Software tools for F-rep Shape Modeling. Computer Graphics & Geometry, vol. 1, No. 10, 1999.

[Bloo94] Bloomenthal, J.: An implicit surface polygonizer, in Graphics Gems IV, P. Heckbert, ed., pp. 324-349, Academic Press, 1994.

[Cerm02a] Čermák, M., Skala, V.: Polygonization by the Edge Spinning, Algoritmy 2002 Conf.proceedings, Univ.of Technology, Slovakia, ISBN 80-227-1750-9, pp.245-252, 2002.

[Cerm02b] Čermák, M., Skala, V.:Edge Spinning Algorithm for Implicit Surfaces, accepted for publications in journal, Mathematics and Computers in Simulation, Elsevier Science, B.V., Roma, Italy.

[Hyp03] HyperFun Project, http://www.hyperfun.org

[Pasko93] Pasko, A., Savchenko, V., Adzhiev, V., Sourin, A.: Multidimensional geometric modeling and vizualization based on the function representation of objects, Technical Report 93-1-008, University of Aizu, Japan, 1993.

[Pasko95] Pasko, A., Adzhiev, V., Sourin, A., Savchenko, V.: Function representation in geometric modeling: concepts, implementation and applications, The Visual Computer, vol.11, No.8, pp.429-446, 1995.

[Rigau99] Rigaudiere,D., Gesquiere,G., Faudot,D.: New Implicit Primitives Used in Reconstruction by Skeletons (France), WSCG, Czech Republic 1999.

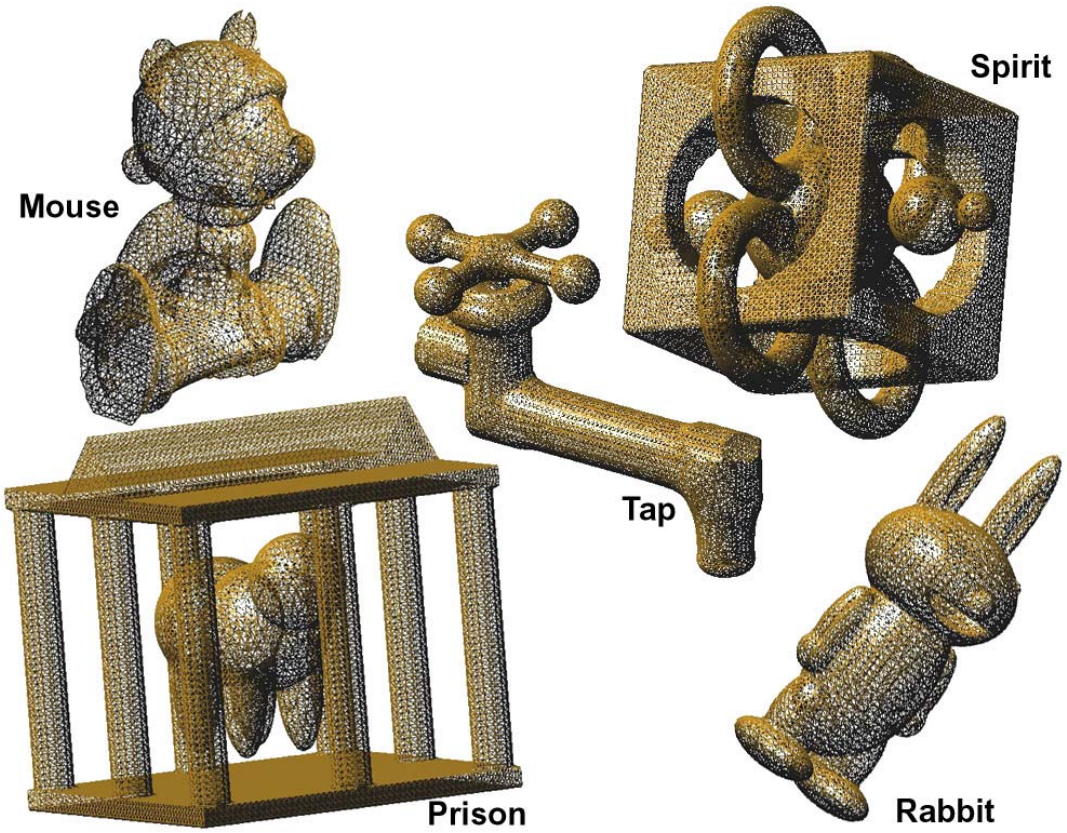

Figure 13. Tested models [Hyp03].